Computer Science 2XC3 - Final Project

Due Date: April 9, 2025 WINTER

Group 9

Wu, Peter: `wu729`

Ganesh, Vikram: `vikramgv`

Bai, Franklin: `baif4`

Contents

Part 1, Team Charter

Communication

For our group, we decided that communication over **Discord** would be the most convenient for all members. Discord also allows screen sharing, making it a valuable tool for collaboration. Each member is expected to respond within **one hour** to maintain efficiency and work within our given timeframe.

For planned video calls, members are expected to join within **15 minutes** of the scheduled time. Since these meetings take up a significant portion of our schedules, punctuality is appreciated. If a member is unable to join, they must notify the group beforehand so the rest can plan accordingly. Any health or family emergencies will, of course, be waivered.

Repeated failure to adhere to the above communication agreements will result in an email to a TA or the professor, outlining the member's inability to meet agreed timeframes. This may lead to a deduction in their grade.

Collaboration Tools

Our team will use **GitHub** to maintain version control. Each member may use the IDE of their choice, provided it is properly synced with Git. A shared repository will be created to store all code, diagrams, and other essential resources. This allows us to collaborate simultaneously, monitor each other's progress, and assist remotely.

The final report will be written using **LateX** to take advantage of its formatting capabilities compared to other programs. Sections may be drafted in other software and later copied or converted into LateX.

Dispute Resolution

Team disputes will be resolved via a discussion on Discord or through a scheduled video call. During this discussion, the team will review the reasons behind any disagreements. If necessary, a second chance may be offered.

If disputes cannot be resolved within the team and repeated failures occur, the matter will be escalated to a TA or the professor via email.

Deadlines and Task Allocation

Our first internal deadline is **March 27th**, during which we will have a video call to discuss our progress on individual components of the project. We will have another internal deadline around **April 7th**, 2 days before the deadline, where we will begin finalizing our project.

Initial task distribution is as follows:

- **Franklin:** Part 1, 6
- **Vik:** Part 2, 3
- **Peter:** Part 4, 5
- **Team:** Part 7

These assignments are not final. Members are encouraged to assist one another, as we recognize the difficulty level may vary between different parts of the project.

Part 2, Single Source Shortest Path Algorithms

Part 3, All-pair Shortest Path Algorithm

Part 4, A* Algorithm

Part 4.1

Issues A_star is trying to address

A_star algorithm is trying to address Dijkstra's algorithm performance problem in a very dense and complex graph, where it has full exploration of the entire graph that is computation heavy. Dijkstra's algorithm also doesn't have the idea of a source going with the destination, which means it would explore a vast number of unnecessary nodes, in the cases when we are not interested in other paths. On the other hand, A_star employs the heuristic function to guide the search towards more promising nodes. This provides an estimate of the cost from the current node to the goal node, allowing A_star to prioritize nodes closer to the goal and cut the unnecessary exploration.

Experiment for Dijkstra's vs A_star

Step 1: Graph Generation
Generate various random graphs with different levels of complexity. Create a set of graphs with the same number of nodes but varying densities. Also, generate another set of graphs with the same number of edges but different sizes (i.e., number of nodes).

Step 2: Keep Track of the Running Time
For each run, ensure that both algorithms are tested on the same graph, with the same randomly selected source node (*src*) and a randomly selected destination node (*dst*) for A_star. After each execution of the algorithm, record the running time.

Step 3: Graph Plotting
We should plot two graphs: both with the y-axis representing the running time. In one plot, the x-axis should represent the number of edges, and in the other, the number of nodes. If desired, include an average trend line across the data points.

Note:
If we are interested in testing different heuristic functions, we can repeat the above

experimental procedure with a different heuristic function. If we are interested in other performance metrics, such as memory usage or the number of nodes explored, we should also record them in Step 2 and plot them as the y-axis variable in additional graphs.

Effects of Arbitrary Heuristic Function

As an arbitrary heuristic is very unlikely to provide any meaningful estimate of the distance to the goal, it would not assist in guiding the search effectively in A_star. The algorithm heavily relies on the admissibility of the heuristic function to make informed decisions. Moreover, a poorly chosen or misleading heuristic can even degrade the performance of the algorithm. In such cases, A_star is no longer guaranteed to find the optimal path, whereas Dijkstra's algorithm still guarantees the shortest path since it does not rely on any heuristic.

Applications of A_star

A_star is particularly feasible in situations where we have an accurate heuristic function. For example, in navigation systems, it is impractical to use Dijkstra's algorithm due to the vast number of possible nodes in a map. However, A_star is much more suitable in this scenario, as it can derive its heuristic function from the straight-line (Euclidean) distance between a node and the destination (dst). In general, for applications where we are only interested in finding the shortest path between two specific nodes, and we have access to a reliable heuristic function, A_star is the preferred choice.

Part 5, Compare Shortest Path Algorithms

Part 6, Organize code per UML Diagram